Chapter 3: Configuration Management

3. 1. Introduction

This chapter describes configuration management (CM) concepts essential to Year 2000 renovations. It is especially important for the Year 2000 effort to ensure that Year 2000 certified code is not inadvertently replaced by an earlier, non-certified version. Many of HUD's application development teams may have adequate procedures to control parallel development. Some of HUD's developers intend to avoid it by managing the Year 2000 activities through sequential releases.

Year 2000 changes will take place over an extended period of time. There is a high probability that new requirements will emerge or unavoidable maintenance activities will surface during this time. It is essential that configuration management plans and policies are instituted and followed to minimize the risks associated with this kind of parallel development.

In this chapter we will cover the fundamentals of configuration management, including HUD's policy for Year 2000 configuration management, and list activities that can be added to current developmental practices to provide continuous improvement.

3. 2. Understanding Configuration Management

When successfully executed, configuration management protects and defends the integrity and quality of information systems being developed or changed. Generally configuration management includes:

- Tools to maintain the integrity of the application system and its components (source, object, JCL, copybooks, etc.);
- A process to record system changes as they occur, and
- An audit trace capability throughout the life of the project.

Although configuration management is perceived as a very meticulous and time consuming process, it safeguards the operational integrity of our applications by administering source and object libraries containing all the components of each business application. It includes the techniques needed to meet the requirements as listed in **Table 3-1**.

Table 3-1: HUD Year 2000 Configuration Management Activities and Requirements Matrix

Requirement	Technique(s)
Manage software migration	Software migration addresses controls that specifically manage movement of code into libraries. Most of the time we think of restricting the unauthorized movement of code into the production library, but software migration may also assure that code is simultaneously promoted to more than one library (e.g., from a staging library to both production and a duplicate production library).
Support back out and recovery	Backout and recovery support the ability to restore the production environment to its previous state, quickly and with confidence.
Enable parallel development through library management	Parallel development requires library management procedures to control access and replacement of source code. It ensures production changes are not inadvertently overwritten because changes were introduced to production between the time the source for a modification was first copied and the modifications were reintroduced. A programmer is assured access to the most recent production version. The process ensures that those with code checked out to them are advised of others who are performing parallel development on that same module. It may, alternatively, deny access to a module's source code until it is checked back in to prevent two programmers from working on the same module concurrently.
Support component identification and version tracking	Component identification and version tracking retain unique references, identifying all of the components of an application, as well as their versions.
Support change control	Change control facilitates being able to identify precisely what changed between events or points in time.

Each development team can readily evaluate the adequacy of their current configuration practices. Indicators that current configuration management practices are adequate include the ability to:

- ➤ Identify the exact set of code used by the application and corresponding to the objects modules actually used;
- ➤ Create a definitive, complete listing of application components.

Under normal circumstances the procedures followed by the development teams are probably adequate to administer the system components and avoid accidentally overlaying a good component with an earlier version or overlooking a component. As the amount of change grows in response to the breadth and depth of Year 2000 activity, we will begin to see the signs of strained resources, the discipline breaking down, the hero forgetting something.

Pragmatically, it is very unlikely that we will be able to completely avoid parallel development (two teams working to introduce different requirements to the same code modules concurrently). This will strain less sophisticated methods of configuration management even further.

The need for strong library management exists when modifications are being made to code concurrently. Developers usually experience this parallel development when emergency fixes are needed while the module is actively being enhanced.

In such a case, the programmer making the enhancements needs to take special measures to incorporate the appropriate fix. Failing to incorporate the emergency fix could result in the enhancement unwittingly reversing out the fix.

3. 3. HUD's Year 2000 Configuration Management Policy

A configuration management process shall be instituted and followed to protect and defend the integrity and quality of information systems developed or maintained during the especially volatile period of Year 2000 renovations. The process shall meet the following requirements:

- **Change control:** the controlled introduction of change to any component; addresses both authorizations and isolation of components to prevent contamination or the introduction of unauthorized change.
- **Status accounting:** the process and procedural methods that enable traceability of a change (e.g., how, when and why did this occur; facilitate backout) and current status for changes either planned or in-process (e.g., who, what, why, when).
- **Component identification:** the identification of the items to be controlled, including techniques to determine generation/versions, and how it was derived.
- **Physical and functional verification:** the review of process controls and techniques that demonstrate the result is derived exclusively from the source (e.g., that the test results were successful, that the test plan and scenarios verified the system requirements, that all of the authorized components and only authorized components were used at any stage).

Table 3-2 (on the following page) shows how specific configuration management activities support these requirements.

3. 4. Recommendations and Suggested Actions

Certain practices should be seriously considered for immediate implementation to facilitate addressing configuration management. (Note: These practices represent a starting set and are not intended to reflect a complete and fully comprehensive configuration management process.)

- Establish Libraries to isolate source and object code. Consider establishing the following libraries:
  - Production source: containing all appropriate source components that were used in generating the production software;
  - Production object: containing the production software;
  - Year 2000 Development: containing the system source components being renovated for Year 2000;
  - Maintenance: containing the system source components requiring change that is independent from, and concurrent with, Year 2000 changes;

Table 3-2: HUD Year 2000 Configuration Management Activities

Configuration Management Activity	Requirement Addressed
1. Establish a production baseline constituting all the necessary and sufficient source code for the application and prohibit write permission to the baseline library except through a single control point. Source code includes JCL/ECL copybooks, procedures, database descriptions, Macros, and so on.	Component Identification
2. Confirm this production baseline is equivalent to the executables in production by compiling the baseline source and conducting a parallel test.	Verification
3. Implement conventions (for example, naming conventions and time date stamps) necessary to be able to identify and track the version of source and executables.	Component Identification and Status Accounting
4. Distribute copies of requested production baseline source module for any development or maintenance from the control point. The control point will make note of the source version and the name of the developer who received the copy.	Status Accounting and Change Control
5. Modify the code and perform appropriate testing in the development libraries by the application developer.	Change Control
6. Return the modified source to the control point when the developer has confirmed testing and is ready for more controlled, rigorous Independent Verification and Validation (IV&V) or acceptance testing.	Change Control
7. Reconcile the modified source with the production baseline source it will replace; avoid accidentally overlaying maintenance fixes. The developer will reconcile and retest software, but the control point must ensure the reconciliation has been done. Tools to facilitate this process include Parallel Development Manager and Endevor on the Hitachi, and basic library and ASCII file compares.	Change Control and Verification
8. Migrate the reconciled source to a separate IV&V or acceptance environment; compile the reconciled source to create executables by the control point.	Change Control
9. Completely test the application to ensure it is production ready; failures are defined to developer, who modifies, tests, and then redelivers the source to the control point. This separate testing ensures the executables were created from the source and that the executables produced the predicted results.	Verification
10. Migrate the modules from the acceptance region to production, (executables go to production and source is migrated to the baseline library—generally this is on the production machine to protect the asset integrity, but as long as there is only one way to write to this library, moving it to something other than the production machine is just as effective for configuration control.)	Change Control and Verification

- Acceptance Y2K Development: containing the system source and production software associated with Year 2000 changes, isolated for system testing;
- Acceptance Maintenance: containing the system source and production software associated with change introduced through the maintenance stream, isolated for system testing;

- Staging to Certification: containing the system source and production software associated with tested Year 2000 changes, ready to undergo future date testing on a fully compliant platform;
- Staging to Production: containing the system source and production software associated with tested Year 2000 changes, ready to be reintroduced into production.

➤ Identify the components to be controlled. Some development teams have done much of this in order to have the system scanned during Impact Assessment. That effort can be used as the base for the initial component identification of the system).

➤ Consistent with HUD's naming conventions, employ naming conventions that would, for example, distinguish between versions; indicate bridges; and designate already renovated software.

➤ Confirm that you have source and objects that created the production object. Recompile source, and relink. Compare the results of identical test scenarios processed against both the current production and the recompiled version.

➤ Establish a single point of contact, a library or release manager, responsible for controlling access to the libraries. All software distribution and reintroduction must pass through this single control point.

➤ Establish specific procedures to prevent accidental introduction of non-compliant code into your environment.

➤ Establish rigorous update procedures for libraries and assign enforcement responsibility. Control of the libraries is absolutely essential for the integrity of the configuration management process.

➤ Describe every change and obtain the proper authorizations to schedule and introduce the change. Emergency changes can follow an expedited version of this procedure.

3. 5. Tools to Facilitate Configuration Management

On the Hitachi, automated configuration management support is provided by Computer Associates' Endevor product. It is not necessary to implement all the capabilities of Endevor to take advantage of some of its features. For example, the developer can employ the library management features for source code without also implementing software migration capabilities of the package.

The Computer Associates' Parallel Development Manager (see the **HUD Year 2000 Tools Overview, Document F** in the **Reference Library**) works with Endevor to support the configuration management process on the Hitachi. Software Quality Assurance (SQA) from Arkdata AB is still being evaluated as an automated configuration management tool for the Unisys platform. On the PC/LAN, Source Safe provides some library management capabilities for source code. PVCS is also a helpful tool in this regard.

3. 6. A Process Overview

The following process does not require the configuration management tools in **Section 3.5** above. It can, however, be supported by those tools.

1. The analyst describes a change.
2. The programmer identifies the components to be included in the release.
3. The librarian delivers the source modules to the programmer following naming conventions that identify the baseline and notifies the programmer of other programmers who have the code checked out.
4. The librarian annotates the library records with the programmer who has it, the release name and the expected release date. The librarian notifies other programmers who have a version of this code checked out to them, that the code has been checked out again, to whom and for what release.
5. The programmer changes and tests modifications to the baseline in accordance with the change requirement. Programmers are to review their changes with other programmers who have the same code checked out to identify and possibly coordinate their activities to minimize functional collisions.
6. The project leader notifies the librarian that the programmer tested software is ready to be moved to the independent testing region (acceptance region).
7. The librarian assures that the software is fully reconciled with the current production version and that the only differences are those introduced by this specific release. Otherwise the programmer performs the necessary reconciliation and returns the software to librarian.
8. The librarian authorizes the recompilation of the software in the acceptance region to assure that the production software has been created from its source.
9. The librarian freezes the acceptance region to prevent interjection of any other source of software change during testing.
10. Programmers, users or other independent teams test the system.
11. Problem reports are referred to the programmers for resolution; authorized software changes are corrected and returned to the librarian (step #6).
12. The librarian is notified and authorized to promote a tested release into the production staging libraries.
13. The librarian moves the appropriate components to the production staging library, including required HUD documentation. The librarian captures source components that created the production version in a source staging library.
14. The librarian notifies the Computer Services Group (CSG) to move the staging libraries into production. When CSG confirms the action, the librarian updates records/naming conventions to record the current production version number, to record that the software has been checked back in with the implementation of the release, and to notify others who have software checked out to them, that a release has been implemented and reconciliation with it will be required.